Cours 420-266-LI **Programmation orientée objet II** **Hiver 2023** **Cégep Limoilou** **Département d'informatique** **Professeur : Martin Simoneau**	**TP2 (18 %)** **Héritage – interface - delegation** **Polymorphisme** **Collection – tableaux multidimensionnels** **Partie ½ (11 %)** **Version - 1**

Objectifs

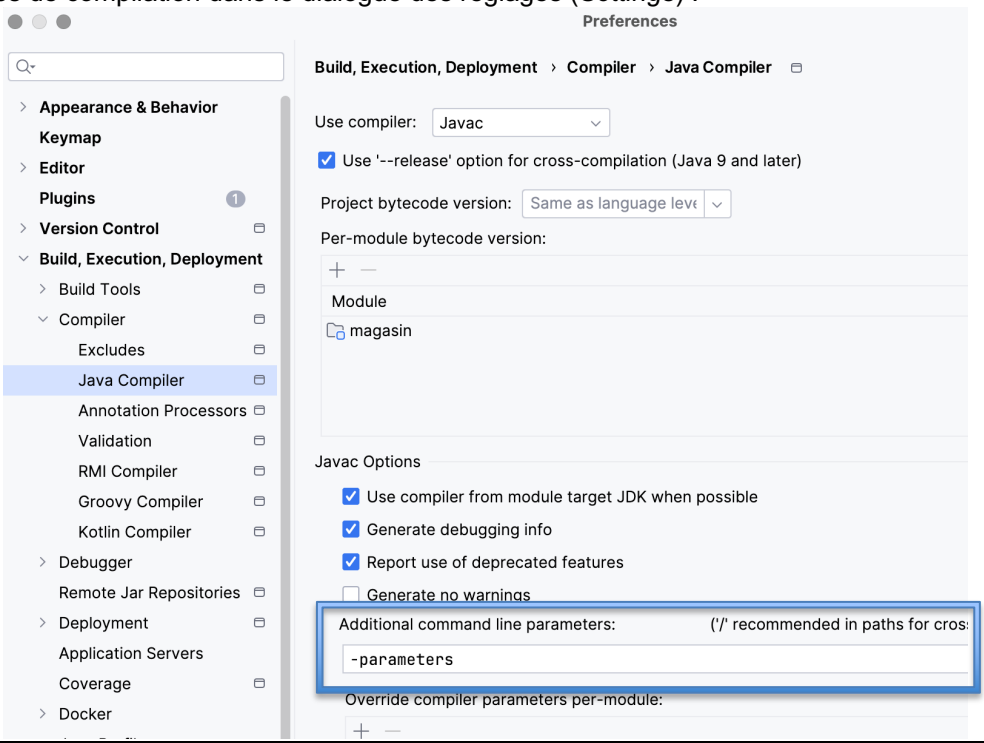
- Réutiliser et rendre compatible en utilisant :
  - Héritage
  - Interface / Abstraction
  - Polymorphisme
  - Tableaux multidimensionnels
  - Collection
    - *List*
    - *Set*
    - *Map*

Contexte :

- Remettre votre projet complet sur Omnivox/*Léa* à la date indiquée.
- Le travail se fait en équipe avec le coéquipier qui vous a été assigné, la copie est interdite.
- Les ajustements de l'énoncés et les mises à jour sur le code seront fournis sur Teams. **Surveillez le canal du tp2.**

Contexte du projet

- Vous allez développer une application permettant de gérer un magasin. Vous aurez à déterminer vous-même les produits qui sont vendus par ce magasin.
- IMPORTANT : Pour que le projet fonctionne. Assurez-vous d'avoir mis le paramètre « ***-parameters*** » dans les paramètres de compilation dans le dialogue des réglages (*Settings*) :



- Vous devez avoir installé le JDK17 de Zulu pour pouvoir faire fonctionner ce TP : https://www.azul.com/downloads/?version=java-17-lts&os=windows&architecture=x86-64-bit&package=jdk-fx - zulu
- On vous a remis un projet avec cet énoncé. Le projet comprend plusieurs package:
    - Le package ***application*** contient tout le code qui lance l'application gérer le UI et interagit avec votre code par l'intermédiaire de l'interface *Modele*. VOUS NE DEVEZ RIEN MODIFIER DANS CE PACKAGE!
    - Le package *client* contient plusieurs sous-package dans lesquels vous aller devoir travailler:
        - ***echange*** contient les interfaces qui permettent au UI d'interagir avec vos classes. Ne rien retirer dans ces interfaces.
        - ***client*** contient les classes relatives au client du magasin: le panier, l'achat et la livraison. Il sera utilisé dans la partie 2.
        - **boite** contient la classe *Boite* qui sert à transférer des produits du fournisseur vers l'entrepôt de votre magasin. Vous pouvez la modifier, mais elle est relativement complète.
        - ***produit*** contient les classes relatives aux produits vendus par le magasin. C'est dans ce package que vous aller concevoir vos classes de produits
        - ***section*** contient les classes relatives aux différents endroits utilisés par le magasin: entrepôt, Vrac, Charite…
    - Pour lancer l'application vous devrez exécuter la méthode main qui se trouve dans la classe *tp2/application/MagasinApplication*.

Partie 1 Choix et création des produits

1. Pour cette partie vous aurez besoin des classes
    a. ***tp2/application/AbstractProduit***
    b. ***tp2/etudiant/Magasin***
    c. ***tp2/etudiant/echange/Descriptible***
2. La première étape consiste à créer les produits que votre magasin peut vendre.
    a. Vous devrez une hiérarchie de classes de produits (ou catégorie de produits).
    b. Chaque équipe doit avoir un type de produit qui lui est propre. Faites valider votre choix par le professeur.
    c. Votre hiérarchie de produits doit avoir **au moins** 3 niveaux et au moins 4 classes concrètes de produits. Les classes abstraites sont nécessaires, mais elles ne comptent pas dans les 4 classes.
        i. Tous vos produits vont être des *AbstractProduit*.
        ii. Chaque classe doit avoir au moins un 1 attribut spécifique (on garde le projet simple). Réutilisez les attributs autant que possible. On se préoccupe du prix et des rabais. Trouvez des attributs qui vont influencer le prix ou le rabais accordé. Les types d'attributs peuvent uniquement être les types suivants:
            1. *String*
            2. *int*
            3. *double*
            4. *boolean*
            5. *float*
            6. *long*
        iii. Vous pouvez ajouter les accesseurs des attributs au besoin.
        iv. Tous les produits ont les attributs suivants:

1. ***Numéro de série***: numéro unique pour chaque exemplaire du produit (initialisation déjà prise en charge par *AbstractProduit)*
2. ***Numéro de produit***: numéro marquant tous les produits du même type (initialisation déjà prise en charge par *AbstractProduit)*
3. ***Date :*** la date de création du produit. (initialisation déjà prise en charge par *AbstractProduit)*
4. ***Nom***: Texte identifiant clairement le produit. Il doit être passé au constructeur de *AbstractProduit*.

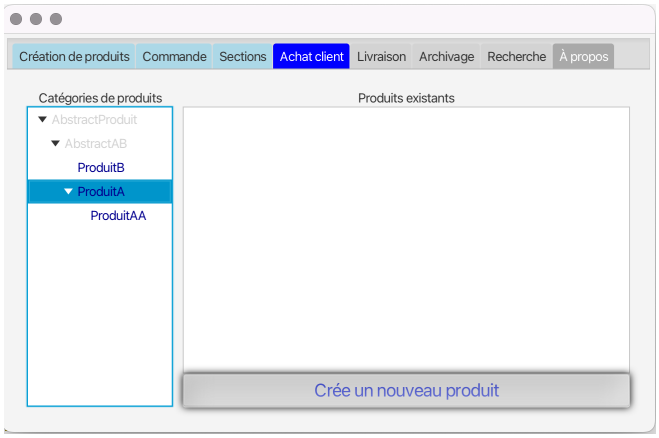
v. Votre classe sera instanciée à partir du premier constructeur défini dans votre classe. Assurez-vous qu'il reçoive tous les attributs nécessaires pour le produit en question.

vi. Notez que les classes de votre hiérarchie représentent des catégories de produits, l'utilisateur de l'application pourra lui-même créer les produits spécifiques. Par exemple, votre application propose comme produit un fruit. L'utilisateur pourra donc créer banane, raison, pomme… simplement en changeant le nom du produit lors de sa création dans le premier onglet de l'application. Ici il n'y a donc pas de classe *Banane*, *Raisin*, *Pomme*. Seules les classes qui ont des attributs différents ont besoin d'être créées.

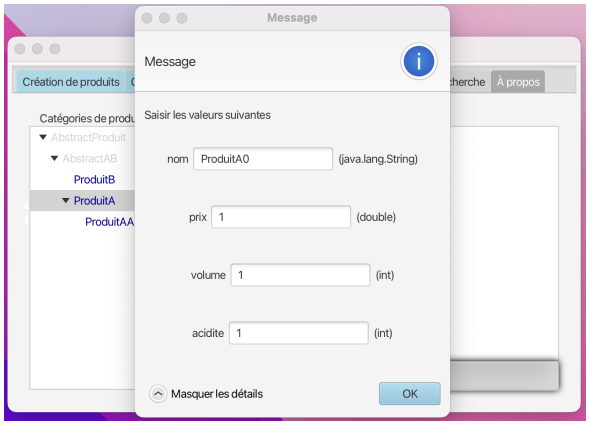
vii. Utilisez cette hiérarchie de produits pour montrer que vous maîtrisez bien les concepts d'héritage et d'abstraction.

viii. Tous les éléments du TP qui doivent être affichés dans l'interface graphique UI, dont les produits, doivent implémenter l'interface ***Descriptible***. Elle possède la méthode ***decrit*** qui retourne une chaîne de caractères. Elle est similaire à *toString*, mais est réservée pour produire un texte simple lisible représentant l'objet. La méthode *toString* génère plutôt une chaîne de caractères représentant l'objet complet. Il peut être utilisé pour diagnostiquer des problèmes dans le débogueur.

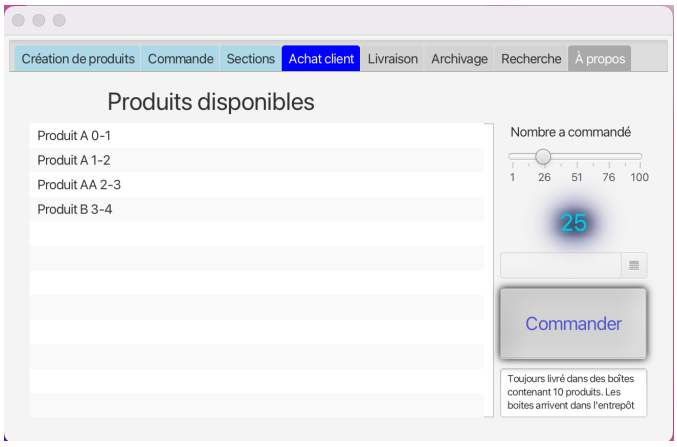
3. Notez que vous pouvez faire tout le reste du TP avec un seul produit. Une personne peut donc travailler sur l'entrepôt (second onglet) pendant que son coéquipier travaille sur les produits. Il suffit de créer un produit bidon temporaire simple qui sera utilisé par le reste de l'application en attendant que les classes de produits soient prêtes.
4. Lorsque vous aurez au moins une classe concrète de produit, l'application va vous la présenter dans l'arbre de catégories de produits du premier onglet. Toutes les classes enfants de *AbstractProduit* qui se trouvent dans le package ***tp2/etudiant/produit*** seront affichées automatiquement.
    a. Lancez l'application et observez la portion gauche de l'onglet *Création de produits*. À gauche, on a les catégories de produits correspondant à chacune de vos classes de produit. La hiérarchie de l'arbre à gauche correspond à votre hiérarchie de classes de produit. Les classes abstraites sont en gris et elles ne peuvent donc pas être utilisées pour créer des produits concrets. La partie de droite montre tous les produits qui ont été créés jusqu'à présent. Ce sont les produits que le magasin pourra commander dans le second onglet. Ici, on montre 3 produits bidons: *ProduitA, ProduitAA* et *ProduitB*.



b. Lorsque l'utilisateur appuie sur le bouton « **Crée un nouveau produit** », il crée alors un produit qu'il pourra commander dans son magasin. L'UI lui présentera un dialogue qui lui permettra de saisir les valeurs précises des attributs pour ce type d'item. Les attributs nécessaires sont déduits à partir des paramètres requis par les constructeurs de vos classes de produits. La figure suivante montre un exemple ou l'on fabrique un *produitA* en lui fournissant les valeurs adéquates. Des valeurs par défaut sont mises dans les champs pour faciliter la création rapide de produit dans les phases de tests.



c. Dans l'onglet ***Commande***, l'utilisateur pourra commander les produits qu'il a créés précédemment. Il lui suffit de cliquer sur un ou plusieurs produits, de choisir la quantité requise avec le curseur (entre 0 et 100) puis d'auppyer sur le bouton ***Commander***.

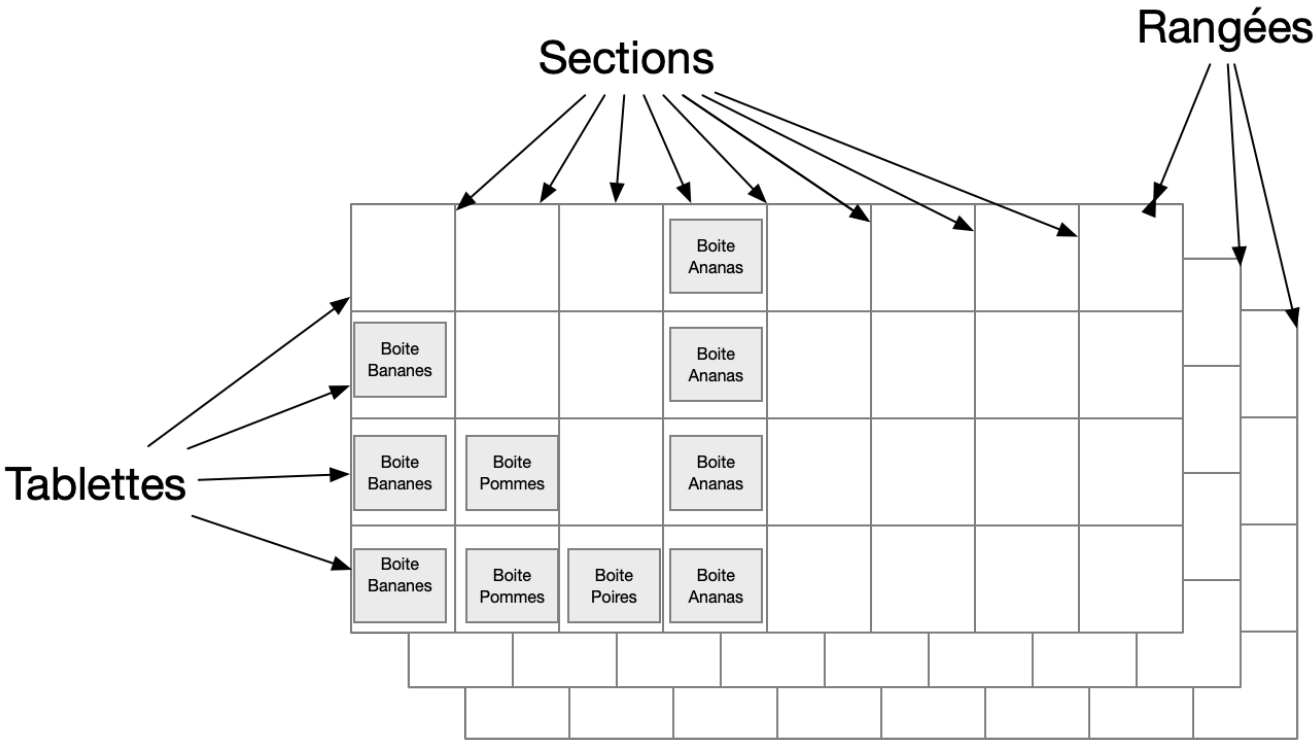


d. Les produits commandés arrivent toujours par boîte de 10. Si la quantité requise n'est pas un multiple de 10, la dernière boîte contiendra moins de 10 éléments.

e. Les items commandés apparaîtront dans l'onglet suivant représentant les sections du magasin.

Partie 2 sections du magasin

1. Pour cette partie vous aurez besoin des classes
   a. ***tp2/etudiant /Boite***
   b. ***tp2/etudiant/Magasin***
   c. ***tp2/etudiant/echange/Descriptible***
   d. ***tp2/etudiant/section/Entrepot***
2. Dans cette partie, vous allez créer les différentes sections du magasin.
   a. D'abord l'entrepôt qui sert à recevoir les boîtes de produits commandés avant de les offrir dans le magasin.
   b. Puis les différentes sections du magasin qui proposent les produits directement au client.
3. Classe **Magasin** :
   a. Cette classe est la classe principale de votre application. Elle implémente l'interface *tp2.echange.Modele* qui inclut toutes les méthodes qui sont nécessaires pour interagir avec l'UI. C'est elle qui contrôle les autres classes de l'application (Entrepot, Vrac, Présentoire, Charite, Panier…).
   b. Méthode de configuration importante pour la première partie
      i. **getAllSection** : permet à l'UI de garder une référence vers vos objets de section de magasin (Présentoires et Vrac) pour pouvoir interagir directement avec eux (présenter du contenu, y placer des produits …).
      ii. **getEntrepot** : permet à l'UI de garder une référence vers votre objet Entrepôt pour pouvoir interagir directement avec lui (présenter leur contenu …).
      iii. **getCharité** : permet à l'UI de garder une référence vers votre objet Charité pour pouvoir interagir directement avec lui (présenter leur contenu
   c. 2 Méthodes importantes de Magasin / Modele pour la première partie.
      i. **recevoirCommande** : Le magasin reçoit les boîtes que l'utilisateur a commandées et il doit les placer dans l'entrepôt. Le magasin est responsable d'envoyer chaque boîte dans l'entrepôt et de retourner le nombre de boîtes qui n'ont pas pu être placées. C'est l'entrepôt qui doit vous indiquer si la boîte peut être reçue. (voir plus loin)
      ii. **placerProduis**: Cette méthode reçoit 2 paramètres : la section qui doit recevoir le produit et les boîtes qui doivent y être transférées. Les boîtes qu'on place dans une section doivent être retirées de l'entrepôt. Si une section ne peut pas recevoir le produit (voir les sections clients plus bas), ce dernier est alors envoyé au Vrac ou à la charité, une classe qui note tout ce que le magasin a donné. Un produit envoyé dans les sections clients ne revient jamais vers l'entrepôt.
4. Classe ***Entrepot***:



entrepot[*rangee*][*section*][*tablette*]

entrepot[categorie][*numéro de produit*][*boite de produit*]

a. La classe *Entrepot* doit contenir un tableau tridimensionnel contenant des objets de type Boite. L'entrepôt possède plusieurs rangées qui contiennent des sections. Chaque section contient des tablettes. La première dimension est le nombre de rangées/étagères et la seconde le nombre d'espaces dans une étagère. Chaque espace de l'étagère ne contient qu'une seule boîte. On a donc un attribut de la forme

   Boite[][][]

b. La première dimension correspond à la catégorie. C'est-à-dire que tous les produits qui sont créés à partir de la même classe se retrouvent dans la même rangée. Les catégories correspondent aux classes concrètes que vous avez programmées dans le package section.

c. La seconde dimension correspond aux différents produits de la catégorie. Ce sont les sections de la rangée. Chaque produit créé par l'utilisateur dans le premier onglet de l'application a un numéro unique. Vous devez trouver une section libre pour placer un nouveau type de produit. Si une section est vidée, elle pourra accueillir un autre type de produit.

d. La dernière dimension correspond aux tablettes de la section. Dans une même section, il n'y a qu'une seule sorte de produit (identifiée par le numéro de produit). Les nouvelles boîtes doivent toujours être placées sur la tablette du bas. Les autres boîtes sont poussées vers le haut. Une section ne doit jamais avoir de trou. Si une boîte est retirée en plein milieu. On doit aussitôt faire descendre les boîtes qui sont au-dessus.

e. La classe *Boite* vous a été fournie. Vous pouvez l'améliorer au besoin, mais si vous la modifiez trop c'est à vos risques !

f. Votre classe *Entrepot* doit avoir une méthode pour ajouter une boîte dans les étagères. Cet algorithme est assez complexe pour que vous ayez à le séparer en plusieurs méthodes. Pour placer une boîte dans une étagère, le magasin suit les règles suivantes:

   i. Une /rangée possède une seule catégorie de produit. Les boîtes reçues ont toujours des objets avec le même numéro de produit et de catégorie. Pour connaître la catégorie d'objet dans la boîte, il suffit d'appeler les méthodes getNumeroCategorie(). Les boîtes ne contiennent qu'une seule sorte de produit (avec le même numéro de produit). Vous devez associer une rangée à chaque catégorie. Cette association peut être fixée à la compilation.

   ii. Un type de produit (*numeroProduit*) n'occupe qu'une seule section. Si la section est pleine, il n'y a plus de place pour ce type de produit dans l'entrepôt. La boîte doit être refusée.

   iii. La boîte la plus récente est toujours placée dans le bas de la section. Lorsqu'une boîte plus récente arrive, les autres boîtes sont poussées vers le haut. Il n'y a jamais de tablette vide entre 2 tablettes contenant des boîtes. Autrement dit, il ne peut y avoir des espaces vides que dans le haut de la section.

   iv. Lorsqu'une boîte arrive et que l'entrepôt n'a pas encore de section allouée pour ce type de produit, il peut la placer n'importe où (idéalement dans le premier espace disponible). L'entrepôt possède un nombre maximal de sections par rangée. Si un nouveau type de produit arrive et qu'il n'y a plus de section disponible pour cette catégorie, le produit doit être refusé (le nombre de produits refusés est retourné par la méthode *recevoirCommande* du *Magasin*).

g. L'entrepôt doit également avoir une méthode pour retirer une boîte. La boîte à retirer est reçue en paramètre. Si on la trouve, on la retire simplement de l'étagère en question. Attention il faut alors boucher le trou en faisant descendre les boîtes qui se trouvaient au-dessus. La section doit rester sans trou.

h. Maintenant que votre *Entrepot* peut recevoir les produits commandés dans l'UI, il reste une étape avant de voir le résultat affiché. Il faut implémenter les 3 méthodes ***getBoites (getBoites1D, getBoites2D, getBoite3D)*** dans la classe *Entrepot* pour que ce dernier puisse indiquer à l'affichage son contenu. Le code que vous avez reçu utilise la méthode getBoites2D, mais il se peut que vous receviez des mises à jour de l'UI qui utilise les 2 autres méthodes.

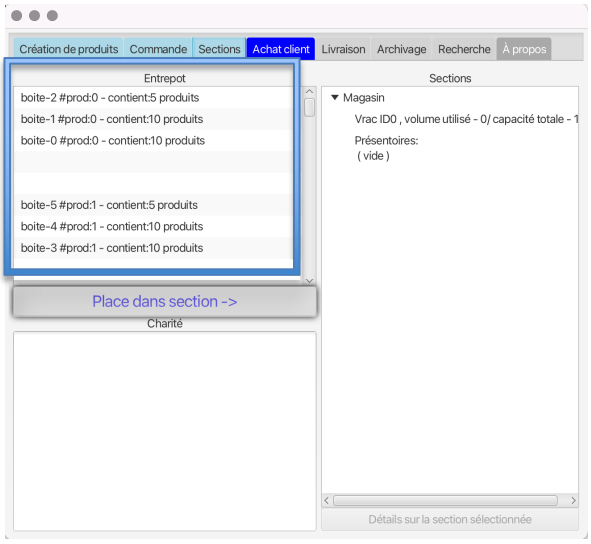
   i. *getBoitesnND* retourne un tableau de *Boite*. Lorsque vous l'aurez implémenté, vous pourrez voir le contenu de votre entrepôt dans l'onglet Sections de l'UI.

      1. *getBoites1D :* retourne toutes les boîtes dans un tableau continu sans trou. C'est utile pour voir le contenu complet de l'entrepôt aisément, mais ça masque la structure et le positionnent des boîtes.

      2. *getBoites2D :* retourne un tableau 2D dans lequel la 2^{e} dimension correspond à la 3^{e} dimension de l'entrepôt (les tablettes de boîtes). Les 2 premières dimensions (catégorie et numeroProduit) doivent être fusionnées. Cette méthode place dans la première dimension toutes les sections, peu importe la catégorie, qui ont au moins une boîte. Dans la seconde dimension, on voit le contenu de chaque section, incluant les tablettes libres (sans boîte).

      3. *getBoites3D* : retourne le tableau tel qu'il est dans l'entrepôt.

   ii. Vous verrez apparaître les produits de l'entrepôt dans la partie gauche de l'onglet ***Sections.***



5. Les **sections clients** *Presentoires (plusieurs Presentoire) et Vrac*
    a. Les sections de magasins doivent être affichées par l'UI. La méthode getAllSection permet à votre Magasin de partager les références vers ses sections à l'UI.
    b. Toutes les sections clients ont un attribut sectionId qui est un nombre qui les identifie de façon unique. Vous devez vous en assurer.
    c. Toutes les sections clients du magasin doivent être des ***Airel***. Une interface utilisée par le UI pour pouvoir afficher le contenu des sections et déplacer des produits. *Airel* contient les méthodes suivantes:
        i. **Collection<AbstractProduit> placerProduits(Boite boite)**
            1. Elle permet de placer le contenu d'une boîte de produit provenant de l'entrepôt directement dans la section. Elle retourne les produits qu'elle n'a pas réussi à placer. Il est important que cette méthode appelle la méthode ***setSectionActuelle*** sur tous les produits reçus afin d'indiquer que le produit est emmagasiné dans cette section.
        ii. **Collection<AbstractProduit> retireProduits(Collection<AbstractProduit> collection)**
            1. Elle permet de retirer les items reçus en paramètre. Elle retourne les produits qu'elle n'a pas réussi à retirer ou à trouver. Elle est principalement utilisée lorsque les produits sont transférés vers le panier du client (TP3).
        iii. **Collection<AbstractProduit> getAllProduits()**
            1. Retourne tous les produits contenus dans la collection.
    d. Vrac
        i. La classe ***Vrac*** est une section client (*Airel*) qui contient plusieurs bacs étiquetés. Chaque bac peut contenir plusieurs produits. On utilisera Une *Map* pour représenter le contenu de bac. Les bacs du vrac sont étiquetés par **nom de produit**. Le contenu du bac est représenté par un Set. Un vrac possédera donc un attribut

            Map<String, Set<AbstractProduit>> contenu;

            Pour ajouter un nouveau produit, on pourra donc utiliser:

contenu.put("nom de produit", new Set<AbstractProduit>());

ii. Pour chaque nom de produit, la map contiendra donc l'ensemble des produits qui ont le même nom.

iii. Lorsque le vrac reçoit un produit :

1. si le nom n'est pas connu dans la *map*, il doit créer un nouveau *set* vide et y ajouter le produit.
2. Si le nom du produit est connu, le nouveau produit sera ajouté dans le *set* déjà associé à son nom

iv. Lorsqu'on retire un produit:

1. Le produit est retiré du *set* et si le *set* est alors vide, il faudra retirer l'entrée complète de la map pour ce nom de produit (clé et valeur).

v. Le *Vrac* ne contient pas de boîtes, on y place les produits directement.

vi. Le Vrac n'a pas de contrainte sur le nombre de produits, mais il a une capacité maximale de volume. C'est-à-dire que la somme des volumes de tous les produits qui s'y trouvent ne doit pas dépasser la capacité maximale du Vrac. Vous devrez probablement ajuster les classes de produits pour gérer le volume qu'ils occupent. Lorsque le vrac est plein, les produits qu'on essaie d'y placer doivent aller automatiquement à la charité.

e. AiresDesPresentoirs

i. La classe ***AireDesPresentoirs*** est une *AireI* qui contient plusieurs objets *Presentoir* (cette classe est présentée plus loin) ordonnés par taille (de celui qui contient le moins de produit à celui qui en contient le plus). Comme tout *AireI*, *AireDesPresentoirs* doit placer et retirer les produits. Il doit aussi transmettre le contenu pour l'affichage.

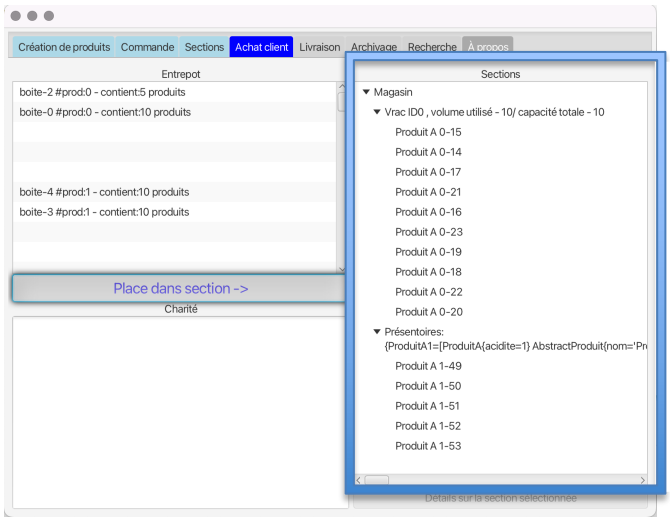
ii. L'aire des présentoirs contient un présentoir pour chaque catégorie de produit. Lorsqu'il reçoit une boîte, celle-ci doit donc être placée dans le bon présentoir. Le présentoir n'est instancié qu'au besoin, lorsqu'un produit de la catégorie est ajouté au magasin.

iii. ***Presentoir*** :

iv. Toutes les instances de *Presentoir* sont gérées par une instance de *AirseDesPresentoire*. Presentoir est une classe qui contient plusieurs espaces de vente. Chaque espace est associé au **nom du produit**. Par exemple, toutes les bananes sont dans le même espace du présentoir. On utilisera Une *Map* pour contenir tous les espaces du présentoir. La clé sera le nom du produit (comme le Vrac), mais la valeur sera une collection capable d'être triée.

1. Map<String, ???> contenu; vous devez choisir le type de ???
2. *Présentoir* ressemble beaucoup au *Vrac*. Vous devrez donc faire attention à bien réutiliser le code!
3. Le présentoir a également une capacité de stockage limitée. Le présentoir à 2 limites :
    a. Il y a un nombre maximum de produits par espace.
    b. Il y a un nombre maximum d'espaces disponibles dans le présentoir.

6. Maintenant, vous devriez pouvoir transférer des boîtes de l'entrepôt principal vers les sections clients en sélectionnant une ou plusieurs boîtes, en sélectionnant une section client cible et en appuyant sur le bouton ***transfert.***



Répartition suggérée pour le travail

Programmeur1:

Entrepot ajoutProduit + *EntrepotClient*

Programmeur 2:

Entrepot + *Vrac* et *Etalage*

Critères d'évaluation et exigences:

Exigences :

- Mettre vos noms en commentaires dans l'en-tête de chacune des classes dans lequel vous avez travaillé.
- Par refactoring, ajouter vos initiales au nom du projet. Exemple : *tp2-etu* devient ***tp2-msd-psg***

  Critères d'évaluation :

Qualité du code :

  - Commentaires pertinents et suffisants. Un développeur normal ne devrait pas mettre plus de 15 secondes pour comprendre ce que fait votre méthode.
  - Une méthode ne devrait pas avoir plus de 20 lignes de code (sans les commentaires)
  - Les noms des classes, méthodes et attributs sont pertinents et **complets**.
  - Il n'y a pas de code inutile ou commenté.
  - Formatage effectué dans chaque classe.
  - Algorithmes simples et efficaces. Toute méthode doit avoir au plus 15 lignes de code sans les commentaires.
  - Le code fonctionne sans erreurs.
  - Toutes les tâches demandées ont été accomplies.

- Classification et polymorphisme

- La réutilisation et la compatibilité sont sans failles.
- L'annotation ***@Override*** est toujours utilisée lorsque c'est possible
- L'abstraction est toujours utilisée lorsqu'elle est pertinente.
- Le code original est respecté (pas de modification inutile du code original, pas de changement de fonctionnalité) lorsque c'est demandé.

FIN